Flow-based generative models

Mathurin Massias

AI hackathon for women in the mathematical sciences

16/02/2026

https://mathurinm.github.io

- Tenured Researcher at INRIA (French institute for Maths & CS)
- PhD in Optimization for ML from Institut Polytechnique de Paris
- Work in ML, Optimization, Generative models
- Part time teacher at Ecole Polytechnique and Ecole Normale Supérieure
- Open source in Python: maintainer of `celer`, `skglm`, `benchopt`

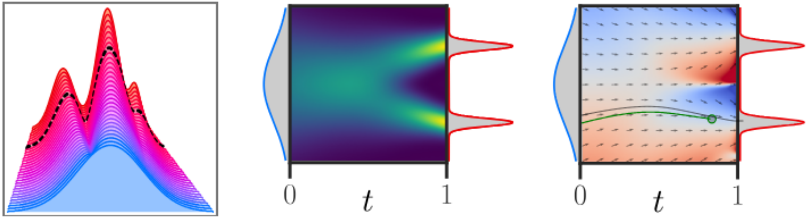
scikit learn   skglm   Benchopt

`https://mathurinm.github.io/`

Blog post on generative models

https://dl.heeere.com/cfm/

“*A Visual Dive into Conditional Flow Matching*”, A. Gagneux, S. Martin, R. Emonet, Q. Bertrand, M. Massias
International Conference on Learning Representations (ICLR) 2025 Blog post



I have tons of additional refs and material, just ask (even after the workshop)

Outline

Generative modelling: the big picture

Normalizing flows

Continuous normalizing flows

Flow matching

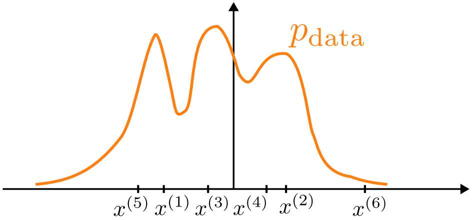
Generative modelling in a nutshell

Given $x^{(1)}, \ldots, x^{(n)}$ sampled from p_{data}, learn to sample from p_{data}

Example:

- $x^{(1)}, \ldots, x^{(n)}$ = real images $\in \mathbb{R}^d$
- p_{data} = distribution of real images

Main challenges of generative modelling?



Generative modelling in a nutshell

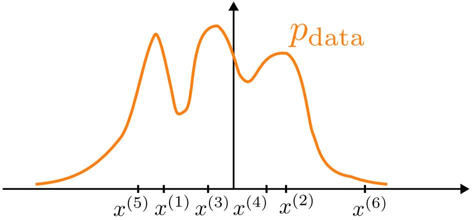
Given $x^{(1)}, \dots, x^{(n)}$ sampled from p_{data}, learn to sample from p_{data}

Example:

- $x^{(1)}, \dots, x^{(n)}$ = real images $\in \mathbb{R}^d$
- p_{data} = distribution of real images

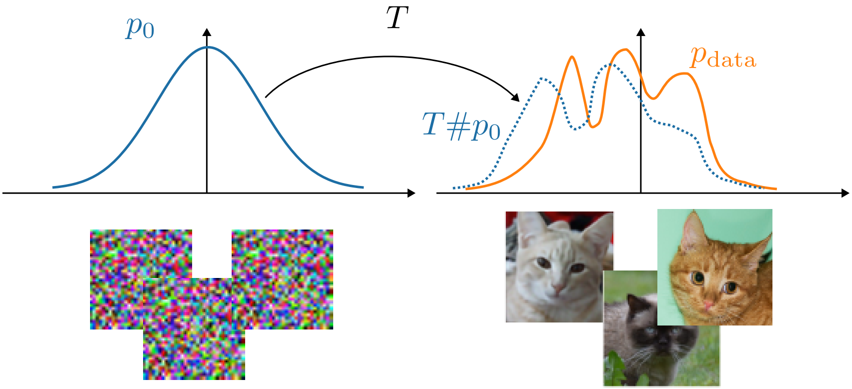
Main challenges of generative modelling?

- enforce fast sampling
- generate high quality samples
- properly cover the diversity of p_{data}



Modern way to do generative modelling

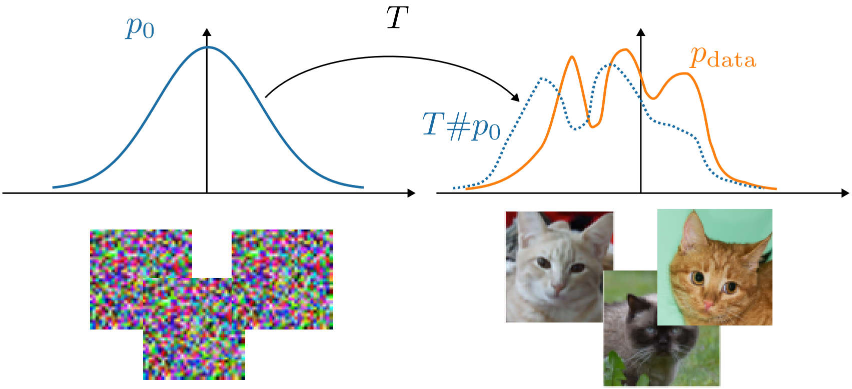
Map **simple** *base distribution* (e.g. Gaussian), p_0, to p_{data} through a map $T : \mathbb{R}^d \to \mathbb{R}^d$



Vocabulary: the distribution of $T(x)$ when $x \sim p_0$ is the *pushforward*, $T\#p_0$

Modern way to do generative modelling

Map **simple** *base distribution* (e.g. Gaussian), p_0, to p_{data} through a map $T : \mathbb{R}^d \to \mathbb{R}^d$



Vocabulary: the distribution of $T(x)$ when $x \sim p_0$ is the *pushforward*, $T\#p_0$

Why should the base distribution be simple?

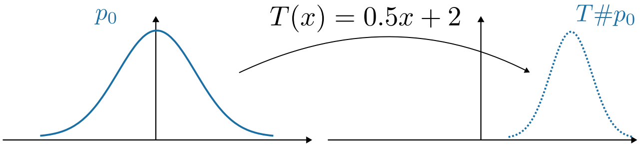
Illustrative example

- In 1D: $x \in \mathbb{R}$
- suppose we only know how to sample from a **standard** Gaussian, $\mathcal{N}(0, 1)$
- we want to generate samples from $\mathcal{N}(a, b^2)$ (Gaussian with mean a, standard deviation b)
- how do we achieve this?

Illustrative example

- In 1D: $x \in \mathbb{R}$
- suppose we only know how to sample from a **standard** Gaussian, $\mathcal{N}(0, 1)$
- we want to generate samples from $\mathcal{N}(a, b^2)$ (Gaussian with mean a, standard deviation b)
- how do we achieve this?

$\hookrightarrow$ we sample x from $\mathcal{N}(0, 1)$, use $T(x) = a + bx$. Then $T(x) \sim \mathcal{N}(a, b^2)$



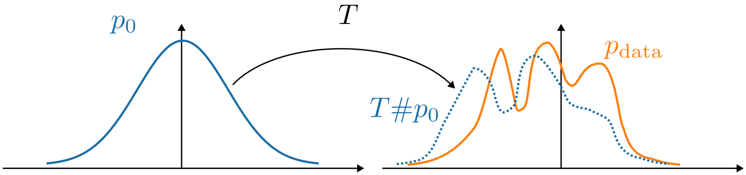
With a more complex T, we can create more complex distributions $T\#p_0$

How to find a good T?

Remember our approach:

- sample x from simple distribution (e.g. Gaussian noise)
- the generated image is $T(x)$

Want: $T\#p_0$ close to p_{data}



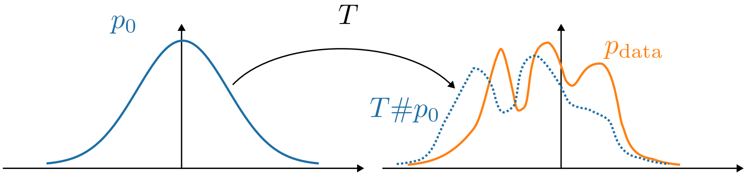
what's the difference with the example in previous slide?

How to find a good T?

Remember our approach:

- sample x from simple distribution (e.g. Gaussian noise)
- the generated image is $T(x)$

Want: $T\#p_0$ close to p_{data}



what's the difference with the example in previous slide?

Big question: "close" in which sense? How could I achieve this?

Outline

Generative modelling: the big picture

Normalizing flows

Continuous normalizing flows

Flow matching

Maximum likelihood detour

- Suppose I flip a coin 10 times, and get: HHTHHTTTHT (5 head, 5 tail)
- Then I ask you to choose between 2 models of the coin:
  - model 1: the coin lands on H with probability 0.1 (on T with proba 0.9)
  - model 2: the coin lands on H with probability 0.5 (on T with proba 0.5)

Which one do you choose? Why?

Maximum likelihood detour

- Suppose I flip a coin 10 times, and get: HHTHHTTTHT (5 head, 5 tail)
- Then I ask you to choose between 2 models of the coin:
  - model 1: the coin lands on H with probability 0.1 (on T with proba 0.9)
  - model 2: the coin lands on H with probability 0.5 (on T with proba 0.5)

Which one do you choose? Why?

- Under model 1, probability of observing said sequence is $0.1^5\,0.9^5 \approx 6.10^{-6}$
- Under model 2, probability of observing said sequence is $0.5^5\,0.5^5 \approx 1.10^{-3}$

"The best model is the one that explains the observed data the best"

Maximum likelihood detour

Is there a model under which the observed sequence is even more probable?
= amongst all models, which is the best?

- suppose you observe n results of a coin toss, $y_1, \ldots, y_n \in \{0, 1\}$
- Bernoulli model: $\mathbb{P}(y = 1) = p \in [0, 1]$ (choosing model = choosing parameter p)
- compact formula $\mathbb{P}(y = y_i) = p^{y_i}(1 - p)^{1-y_i} \in [0, 1]$
- for a given p, what is the probability of observing the full observation set $(y_1, \ldots, y_n)$?

Maximum likelihood detour

Is there a model under which the observed sequence is even more probable?
= amongst all models, which is the best?

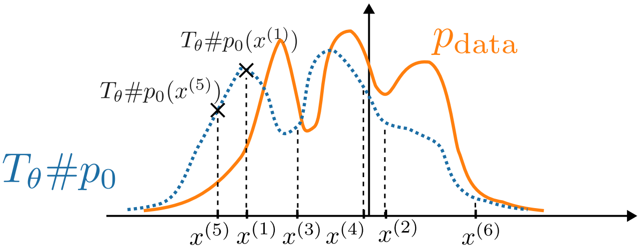
- suppose you observe n results of a coin toss, $y_1, \ldots, y_n \in \{0, 1\}$
- Bernoulli model: $\mathbb{P}(y = 1) = p \in [0, 1]$ (choosing model = choosing parameter p)
- compact formula $\mathbb{P}(y = y_i) = p^{y_i}(1-p)^{1-y_i} \in [0, 1]$
- for a given p, what is the probability of observing the full observation set $(y_1, \ldots, y_n)$?
- *likelihood* of the observations (probability to observe $(y_1, \ldots, y_n)$): $\prod_1^n p^{y_i}(1-p)^{1-y_i}$
- maximize the likelihood $\iff$ minimize the negative log likelihood $\iff$ $\min_p - \sum_1^n y_i \log p - \sum_1^n (1 - y_i) \log(1 - p)$
- solution in p?

Back to generative: how to find a good T

- choose T as parametric map: T_θ (a neural network)
- find best parameters θ by **maximizing the log-likelihood** of available samples:

$$\theta^* = \operatorname*{argmax}_\theta \sum_{i=1}^n \log \Big(\underbrace{(T_\theta \# p_0)}_{:= p_1}(x^{(i)}) \Big)$$

(links with empirically minimizing the Kullback-Leibler divergence $\mathrm{KL}(p_{\mathrm{data}}, T_\theta \# p_0)$)
`https://mathurinm.github.io/blog/kl_mle/`



How to find a good T: computing the likelihood

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \log\left(\underbrace{(T_\theta \# p_0)}_{:= p_1}(x^{(i)})\right)$$

- we have this objective to maximize in θ, but can we actually compute it?

How to find a good T: computing the likelihood

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \log \left(\underbrace{(T_\theta \# p_0)}_{:= p_1}(x^{(i)}) \right)$$

- we have this objective to maximize in θ, but can we actually compute it?
- we can rely on the *change of variable formula*:

$$\log p_1(x) = \log p_0(T_\theta^{-1}(x)) + \log |\det J_{T_\theta^{-1}}(x)|$$

How to find a good T: computing the likelihood

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \log \Big(\underbrace{(T_\theta \# p_0)}_{:= p_1}(x^{(i)}) \Big)$$

- we have this objective to maximize in θ, but can we actually compute it?
- we can rely on the *change of variable formula*:

$$\log p_1(x) = \log p_0(T_\theta^{-1}(x)) + \log |\det J_{T_\theta^{-1}}(x)|$$

$J_{T_\theta^{-1}}$ is the *Jacobian* (=matrix of partial derivatives – in 1D: $J_f(x) = f'(x)$)

Exercise: $p_0 = \mathcal{N}(0, 1)$, $T_\theta(x) = ax + b$, compute T_θ^{-1}, its derivative, and then p_1

The change of variable formula

$$\boxed{\log p_1(x) = \log p_0(T_\theta^{-1}(x)) + \log|\det J_{T_\theta^{-1}}(x)|}$$

= a mathematical formula to compute the probability of a generated image $T_\theta(x)$

In practice we'll use a neural network for T_θ. What do we need?

The change of variable formula

$$\log p_1(x) = \log p_0(T_\theta^{-1}(x)) + \log|\det J_{T_\theta^{-1}}(x)|$$

= a mathematical formula to compute the probability of a generated image $T_\theta(x)$

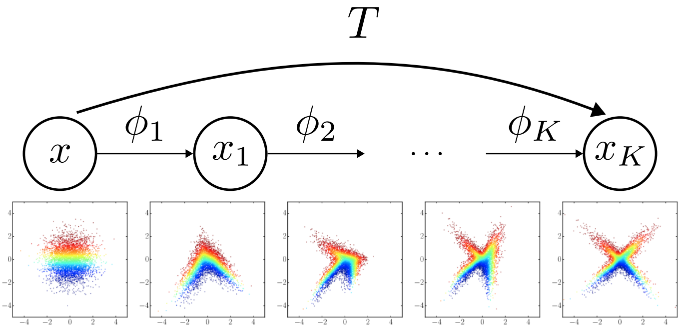
In practice we'll use a neural network for T_θ. What do we need?

- T_θ must be invertible
- T_θ^{-1} should be easy to compute in order to evaluate the first right-hand side term
- T_θ^{-1} must be differentiable
- the (log) determinant of the Jacobian of T_θ^{-1} must not be too costly to compute

Normalizing Flows (2015) = neural architectures satisfying these requirements

Normalizing flows

- Key observation: If T and T' satisfy the requirements, so does $T \circ T'$
- Build T as composition of simple blocks ϕ_k satisfying the invertibility + Jacobian constraints



(picture from Rezende & Mohamed 2015)

Examples of normalizing flows

- planar flow: $\phi_k(x) = x + \sigma(b_k^\top x + c_k)a_k$ (parameters to learn $a_k \in \mathbb{R}^d, b_k \in \mathbb{R}^d, c_k \in \mathbb{R}$)

$$J_{\phi_k}(x) = \mathrm{Id} + \sigma'(b_k^\top x + c_k)a_k b_k^\top$$

id + rank one, all good for the determinant ($\det(\mathrm{Id} + uv^\top) = 1 + v^\top u$)

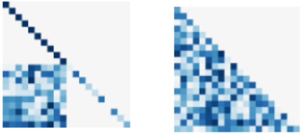
Examples of normalizing flows

- planar flow: $\phi_k(x) = x + \sigma(b_k^\top x + c_k)a_k$ (parameters to learn $a_k \in \mathbb{R}^d, b_k \in \mathbb{R}^d, c_k \in \mathbb{R}$)

$$J_{\phi_k}(x) = \mathrm{Id} + \sigma'(b_k^\top x + c_k)a_k b_k^\top$$

  id + rank one, all good for the determinant ($\det(\mathrm{Id} + uv^\top) = 1 + v^\top u$)

- real NVP (triangular Jacobian, details in blog post)



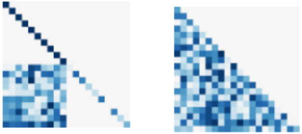
Examples of normalizing flows

- planar flow: $\phi_k(x) = x + \sigma(b_k^\top x + c_k)a_k$ (parameters to learn $a_k \in \mathbb{R}^d, b_k \in \mathbb{R}^d, c_k \in \mathbb{R}$)

$$J_{\phi_k}(x) = \mathrm{Id} + \sigma'(b_k^\top x + c_k)a_k b_k^\top$$

  id + rank one, all good for the determinant ($\det(\mathrm{Id} + uv^\top) = 1 + v^\top u$)

- real NVP (triangular Jacobian, details in blog post)



but **too many constraints** on the architecture, restricts the expressivity

Outline

Generative modelling: the big picture

Normalizing flows

Continuous normalizing flows

Flow matching

From discrete to continuous time: ResNets

Residual Networks (ResNets): from layer ℓ equation

$$x_{\ell+1} = \sigma(W x_\ell + b_\ell)$$

... to

$$x_{\ell+1} = x_\ell + \sigma(W x_\ell + b_\ell)$$

Why does this help?

From discrete to continuous time: ResNets

Residual Networks (ResNets): from layer ℓ equation

$$x_{\ell+1} = \sigma(W x_\ell + b_\ell)$$

... to

$$x_{\ell+1} = x_\ell + \sigma(W x_\ell + b_\ell)$$

Why does this help?

Continuous time limit: Neural Ordinary Differential Equations

$$\begin{aligned} x_{\ell+1} &= x_\ell + \delta\sigma(W x_\ell + b_\ell) \\ \frac{x_{\ell+1} - x_\ell}{\delta} &= \sigma(W x_\ell + b_\ell) \\ &:= u_\ell(x_\ell) \end{aligned}$$

Is the last equation reminiscent of something?

From discrete to continuous time: ResNets

Residual Networks (ResNets): from layer ℓ equation

$$x_{\ell+1} = \sigma(W x_\ell + b_\ell)$$

... to

$$x_{\ell+1} = x_\ell + \sigma(W x_\ell + b_\ell)$$

Why does this help?

Continuous time limit: Neural Ordinary Differential Equations

$$\begin{aligned} x_{\ell+1} &= x_\ell + \delta \sigma(W x_\ell + b_\ell) \\ \frac{x_{\ell+1} - x_\ell}{\delta} &= \sigma(W x_\ell + b_\ell) \\ &:= u_\ell(x_\ell) \end{aligned}$$

Is the last equation reminiscent of something?

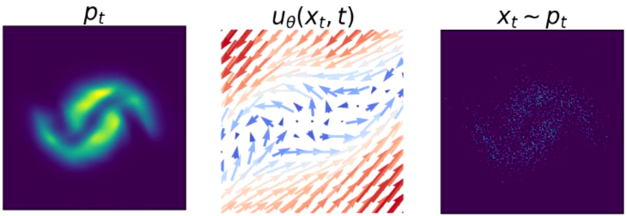
$$\partial_t x(t) = u(x(t), t)$$

Continuous normalizing flows

- define T_θ **implicitly** through ODE: $T_\theta(x_0) := x(1)$, where

$$\begin{cases} x(0) = x_0 \\ \partial_t x(t) = u_\theta(x(t), t) \quad \forall t \in [0,1] \end{cases}$$

- learn the **velocity field** $u_\theta : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$

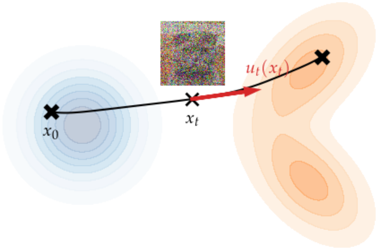


(dynamic animation in blog post)

First win: the mapping defined by the ODE, $T(x_0) := x(1)$ is inherently invertible (why?)

Recap: continuous normalizing flows (CNF)

- work in the continuous-time domain: $t \in [0, 1]$
- model the continuous solution $(x(t))_{t \in [0,1]}$
- learn the **velocity field** u as $u_\theta : \mathbb{R}^d \times [0, 1] \to \mathbb{R}^d$
- sample by solving the ODE with $x_0 \sim p_0$



The map T is no longer explicit, it is defined by solving an ODE

Mathematical toolbox: the IVP trifecta

$$\begin{cases} x(0) = x_0 \\ \partial_t x(t) = u(x(t), t) \quad \forall t \in [0,1] \end{cases}$$

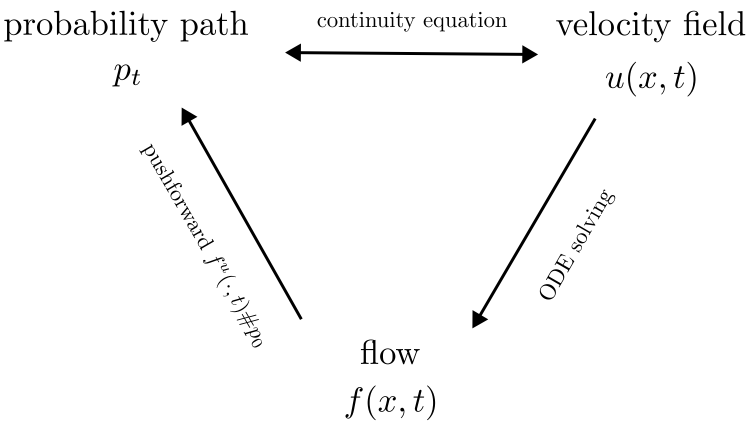
3 objects associated to this ODE:

- the **velocity field** $u : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$
- the **flow** $f^u : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$: $f^u(x,t)$ = solution at time t to the initial value problem with initial condition $x(0) = x$
- the **probability path** $(p_t)_{t\in[0,1]}$ = the distributions of $f^u(x,t)$ when $x \sim p_0$ ($p_t = f^u(\cdot, t) \# p_0$)

Link: continuity equation

$$\boxed{\partial_t p_t + \mathrm{div}(u_t p_t) = 0}$$

The IVP trifecta



How do we learn the velocity u_θ now?

- Saved by the *instantaneous* change of variable formula:

$$\frac{\mathrm{d}}{\mathrm{d}t} \log p_t(x(t)) = -\mathrm{tr}\, J_{u_\theta(\cdot,t)}(x(t)) = -\operatorname{div} u_\theta(\cdot, t)(x(t)) \quad \forall t \in [0,1]$$

- allows computing $\log p_1(x^{(i)})$: solving ODE
- nice: avoid computing the full Jacobian with the Hutchinson trace trick (`https://mathurinm.github.io/blog/hutchinson/`)
- constraints on u much less stringent than in discrete normalizing flows: only need unique ODE solution (OK if u Lipschitz in x and continuous in t)

Issues of CNFs

- during training, we need to solve ODEs (why?)
- we then need to backpropagate inside an ODE solver $\hookrightarrow$ no black box
- this is terribly unstable

$\hookrightarrow$ Flow Matching solves this: a different way to train CNFs!

Outline

Generative modelling: the big picture

Normalizing flows

Continuous normalizing flows

Flow matching

Recap

We have:

- source distribution $p_0 = \mathcal{N}(0, \mathrm{Id})$
- target distribution p_{data} (e.g. realistic images)

We want:

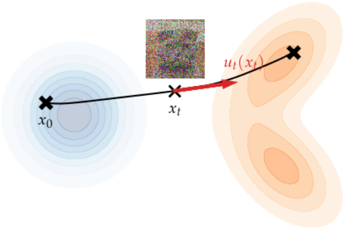
- to generate new samples from p_{data}

How?

- by solving on $[0, 1]$

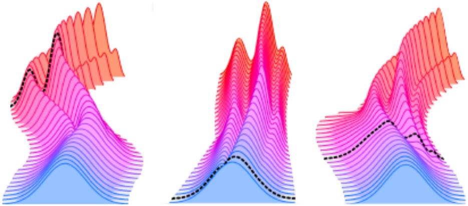
$$\begin{cases} x(0) = x_0 \\ \dot{x}(t) = u(x(t), t) \quad \forall t \in [0, 1] \end{cases}$$

- such that solution $x(1) \sim p_{\mathrm{data}}$ when $x(0) \sim p_0$



Searching for a good u

$$\begin{cases} x(0) = x_0 \\ \dot{x}(t) = u(x(t), t) \quad \forall t \in [0, 1] \end{cases}$$



- ODE defines *probability path* $(p_t)_{t\in[0,1]}$ = laws of the solution $x(t)$ when $x(0) \sim p_0$
- many ways to go from p_0 to $p_1 = p_{\text{data}}$

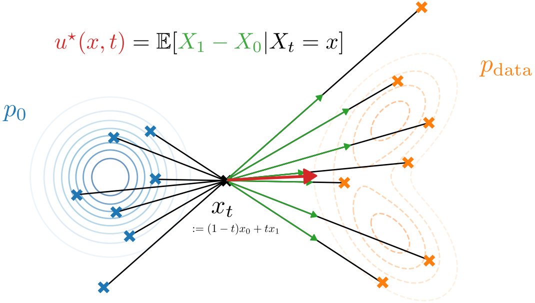
Flow matching targets a specific probability path/velocity

Searching for a good u: the magic

Theorem 1

Define $X_t \triangleq (1-t)X_0 + tX_1$ (X_0: noise, X_1: clean image). Then:

$$u^\star(x,t) := \mathbb{E}[X_1 - X_0 | X_t = x] \text{ transports } p_0 \text{ to } p_{\text{data}}$$



Proof: 4 lines, based on continuity equation.

We are done

- we have our target, valid velocity:

$$u^\star(x,t) = \mathbb{E}[X_1 - X_0 | X_t = x]$$

- L2 characterization of conditional expectation:

$$\mathbb{E}[Y|Z = \cdot] = \underset{f \text{ measurable}}{\operatorname{argmin}} \ \mathbb{E}_{Y,Z}\|Y - f(Z)\|^2$$

- so we can approximate $u^\star$ with a neural network u_θ, by solving:

$$\boxed{\min_\theta \mathbb{E}_{\substack{x_0 \sim p_0 \\ x_1 \sim p_{\text{data}} \\ t \sim \mathcal{U}([0,1])}} \|u_\theta(x_t, t) - (x_1 - x_0)\|^2}$$

where $x_t := (1-t)x_0 + tx_1$

- why are we happy with this training loss?

Training flow matching

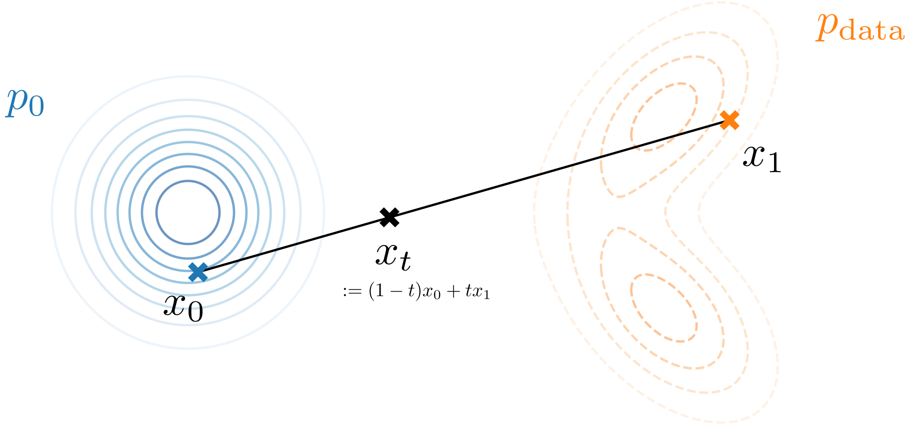
$$\min_{\theta} \mathbb{E}_{\substack{x_0 \sim p_0 \\ x_1 \sim p_{\text{data}} \\ t \sim \mathcal{U}([0,1])}} \left[\| u_\theta(x_t, t) - (x_1 - x_0) \|^2 \right] \qquad x_t := (1-t)x_0 + t x_1$$

Training flow matching

$$\min_{\theta} \mathbb{E}_{\substack{x_0 \sim p_0 \\ x_1 \sim p_{\text{data}} \\ t \sim \mathcal{U}([0,1])}} \left[\| u_\theta(x_t, t) - (x_1 - x_0) \|^2 \right] \qquad x_t := (1-t)x_0 + t x_1$$

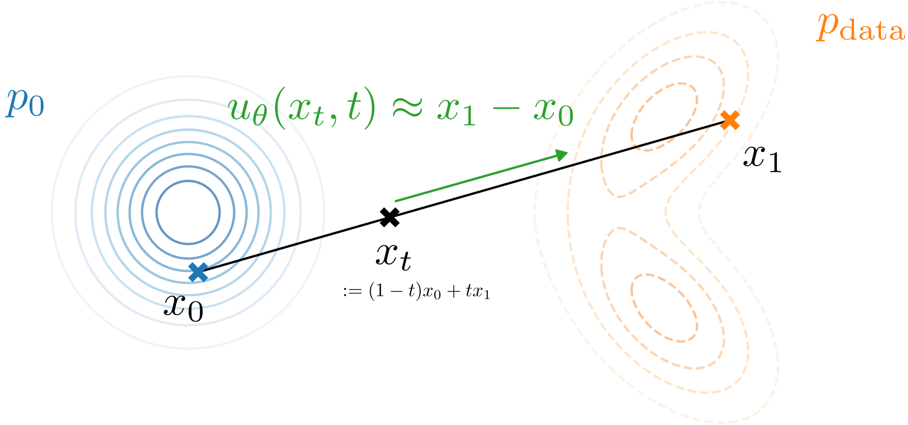
Training flow matching

$$\min_{\theta} \mathbb{E}_{\substack{x_0 \sim p_0 \\ x_1 \sim p_{\text{data}} \\ t \sim \mathcal{U}([0,1])}} \left[\| u_\theta(x_t, t) - (x_1 - x_0) \|^2 \right] \qquad x_t := (1-t)x_0 + t x_1$$



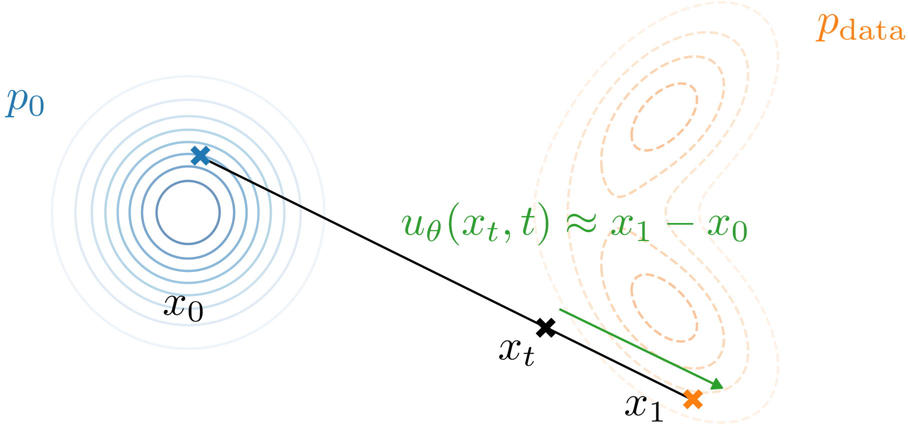
Training flow matching

$$\min_{\theta} \mathbb{E}_{\substack{x_0 \sim p_0 \\ x_1 \sim p_{\text{data}} \\ t \sim \mathcal{U}([0,1])}} \left[\| u_\theta(x_t, t) - (x_1 - x_0) \|^2 \right] \qquad x_t := (1-t)x_0 + t x_1$$



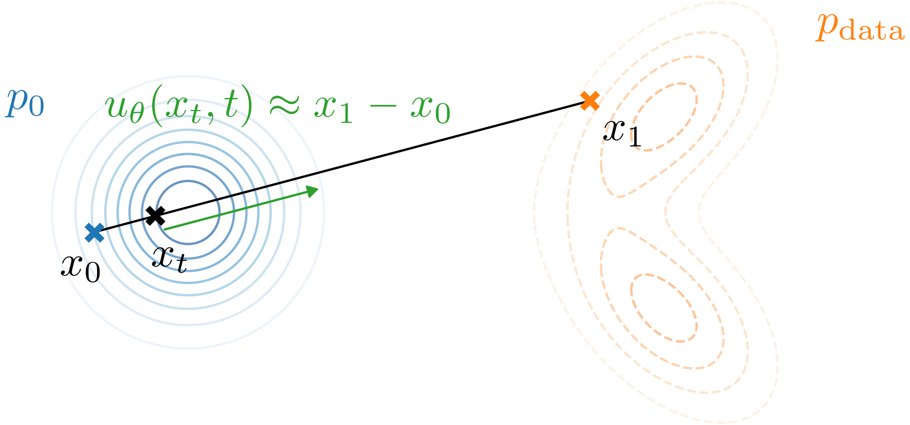
Training flow matching

$$\min_{\theta} \mathbb{E}_{\substack{x_0 \sim p_0 \\ x_1 \sim p_{\text{data}} \\ t \sim \mathcal{U}([0,1])}} \left[\| u_\theta(x_t, t) - (x_1 - x_0) \|^2 \right] \qquad x_t := (1-t)x_0 + t x_1$$



Training flow matching

$$\min_{\theta} \mathbb{E}_{\substack{x_0 \sim p_0 \\ x_1 \sim p_{\text{data}} \\ t \sim \mathcal{U}([0,1])}} \left[\| u_\theta(x_t, t) - (x_1 - x_0) \|^2 \right] \qquad x_t := (1-t)x_0 + t x_1$$



Additional topics that we can discuss

- why does flow matching create new data?
- *discrete* flow matching
- equivalence with diffusion
- conditioning on text (prompt)
- autoregressive models (GPT-like)

Diffusion lab: `https://github.com/Badr-MOUFAD/gen-ai-lab1/`

Notebook time

`https://mathurinm.github.io/teaching/`

- `lab_fm_full.py`: click and play
- `lab_fm_mid.py`: fill training loop
- `lab_fm_todo.py`: fill generation, training, plots